From UML to Process Algebra and Back:

An Automated Approach to Model-Checking Software Design Artifacts of Concurrent Systems

Daniela Remenska, Jeff Templon , Tim A.C. Willemse,

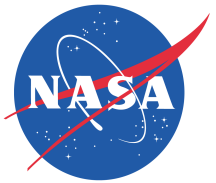
Philip Homburg, Kees Verstoep , Adria Casajus , and Henri Bal

NIKHEF

VU VRIJE UNIVERSITEIT AMSTERDAM

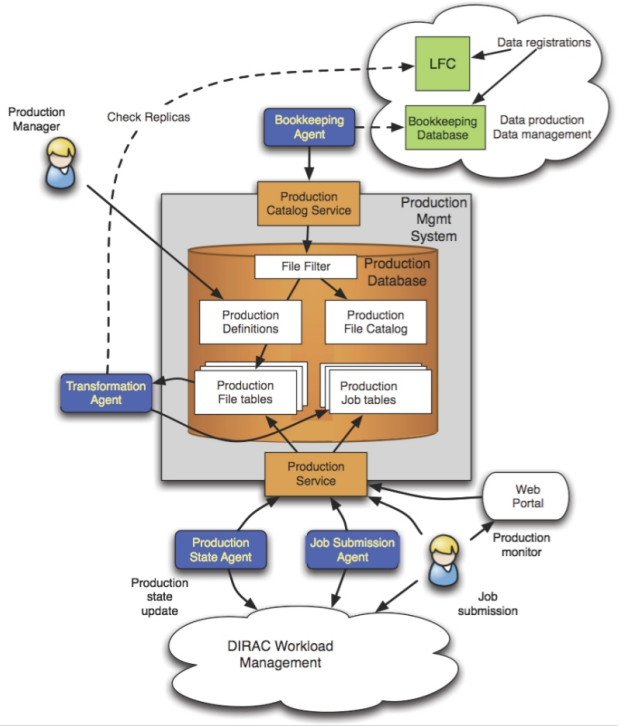
TU/e Technische Universiteit Eindhoven University of Technology

Universitat de Barcelona



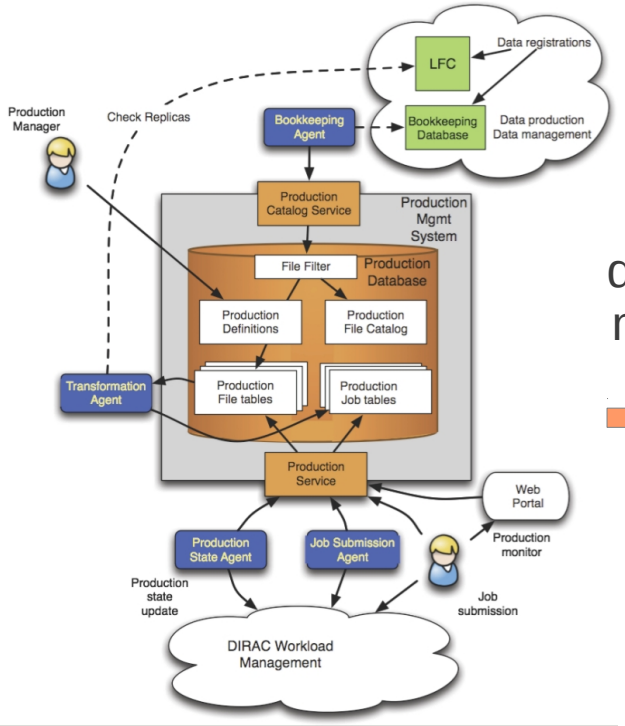
NASA Formal Methods Symposium
May 2013

Model checking: a success story?



Software/Hardware System

Model checking: a success story?

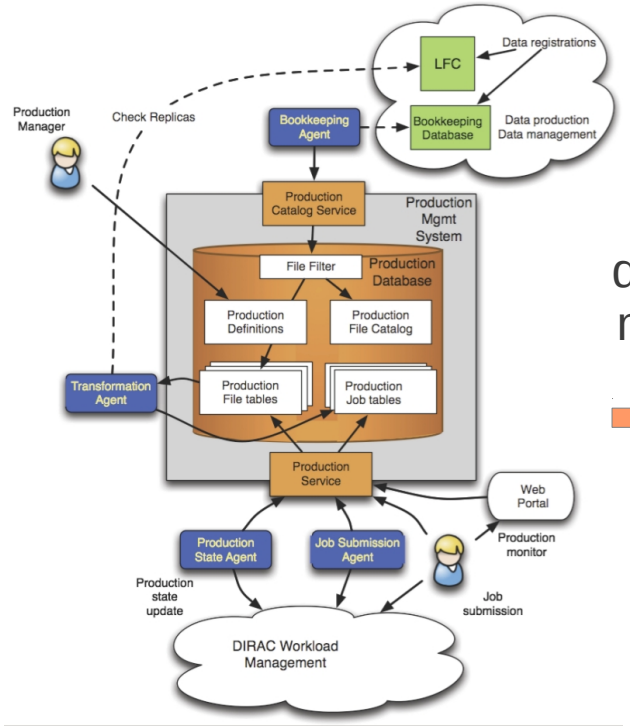


Software/Hardware System

Are there deadlocks in my system?

Model Checker

Model checking: a success story?



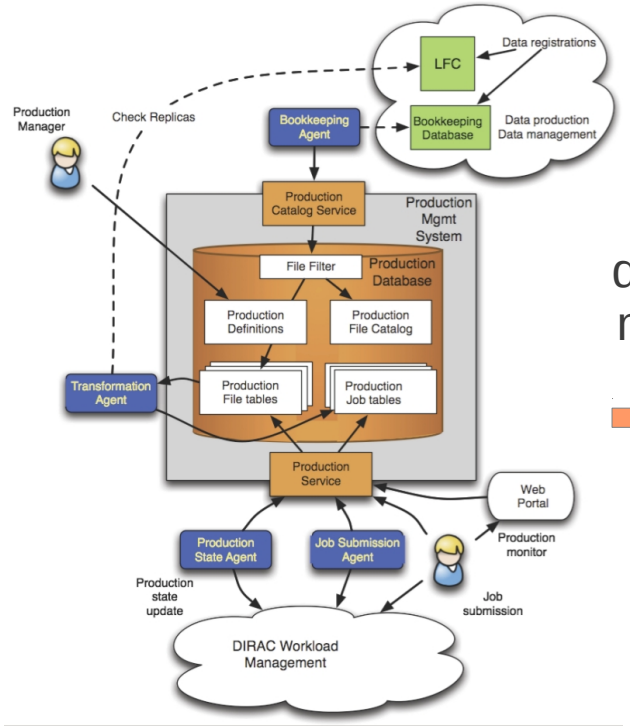
Software/Hardware System

Are there deadlocks in my system?

Model Checker

Nope, you're good to go!

Model checking: a success story?



Software/Hardware System

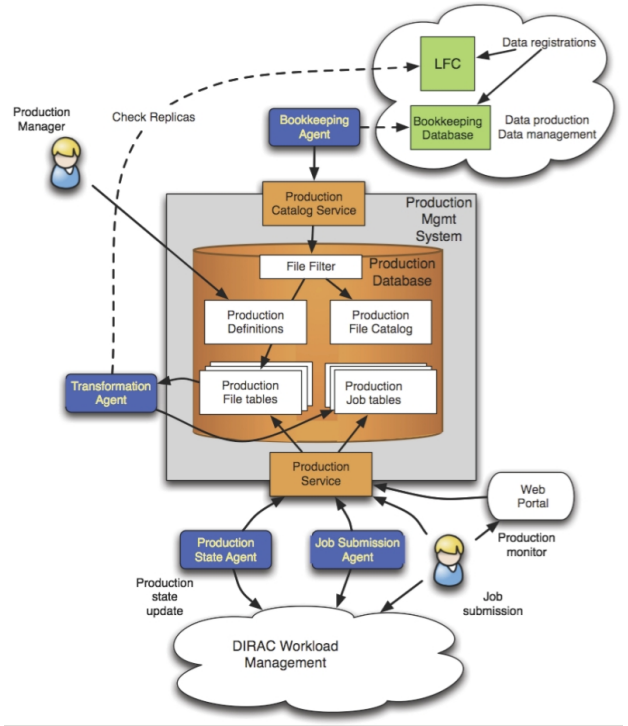
Are there deadlocks in my system?

Model Checker

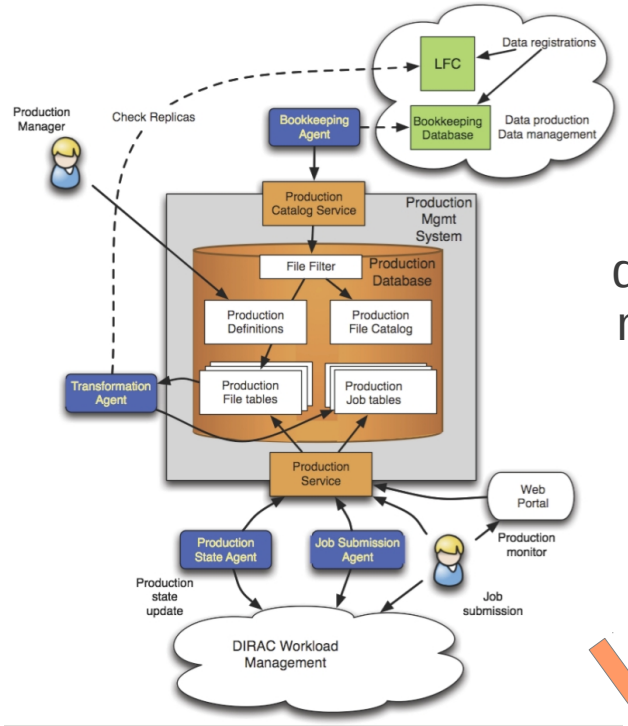
Nope, you're good to go!

Houston, we have a problem!

Model checking: a success story?



Model checking: a success story?



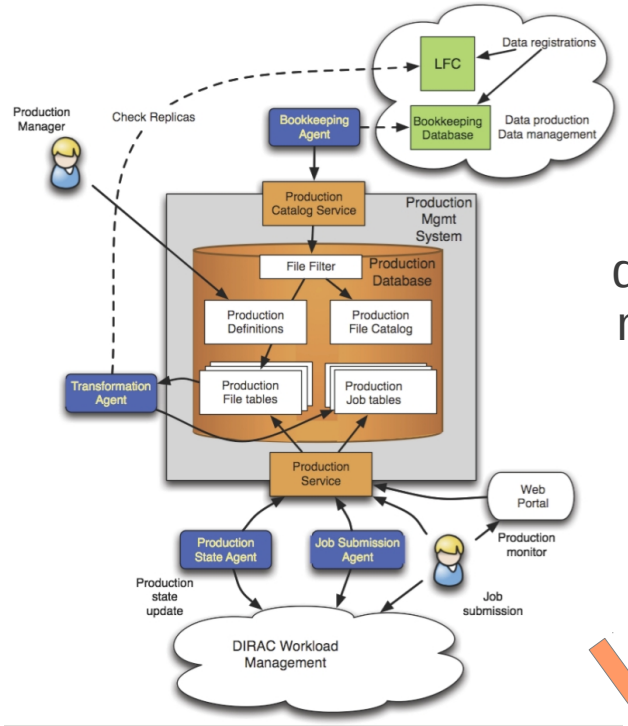
Are there deadlocks in my system?

Model Checker

Software/Hardware System

```
P_req(pid:Pos,t:Time,x:Time)=
  sum u:Time.(u<k)-> set(pid)@(t+u).P wait(pid,t+u,0);
P_wait(pid:Pos,t:Time,x:Time)=
  sum u:Time.(x+u>=k)->_get(pid)@(t+u).cs_in(pid)@(t+u).P_cs(pid,t+u,x+u)
 +sum u:Time. get(0)@(t+u).P req(pid,t+u,0);
```

Model checking: a success story?



Are there deadlocks in my system?

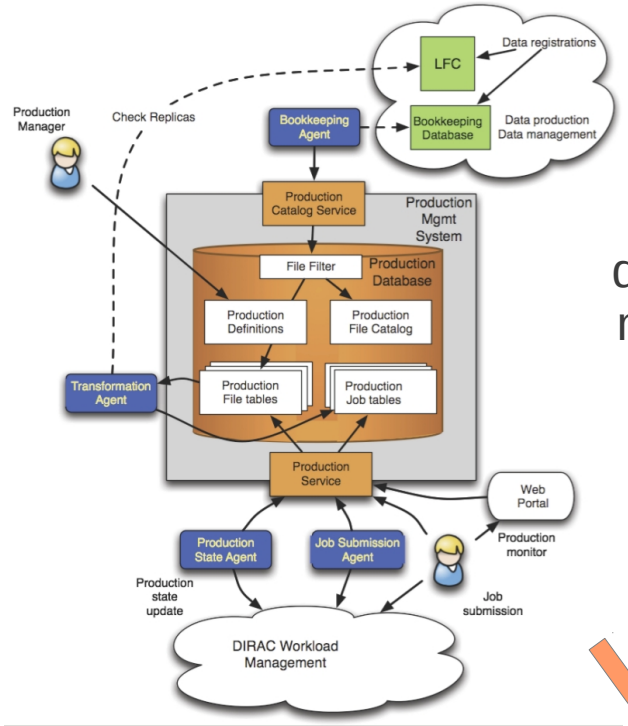
Model Checker

Software/Hardware System

```
P_req(pid:Pos,t:Time,x:Time)=
  sum u:Time.(u<k)-> set(pid)@(t+u).P wait(pid,t+u,0);
P_wait(pid:Pos,t:Time,x:Time)=
  sum u:Time.(x+u>=k)->_get(pid)@(t+u).cs_in(pid)@(t+u).P_cs(pid,t+u,x+u)
 +sum u:Time. get(0)@(t+u).P req(pid,t+u,0);
```

```
<true*>(exists d:D .<ra(d)>
     (nu X. mu Y. (<i_lost>X || <!i_lost && !sb(d)>Y)))
```

Model checking: a success story?



Software/Hardware System

Are there deadlocks in my system?

Model Checker

Nope, you're good to go!

Houston, we have a problem!

```
P_req(pid:Pos,t:Time,x:Time)=
  sum u:Time.(u<k)-> set(pid)@(t+u).P wait(pid,t+u,0);
P_wait(pid:Pos,t:Time,x:Time)=
  sum u:Time.(x+u>=k)->_get(pid)@(t+u).cs_in(pid)@(t+u).P_cs(pid,t+u,x+u)
 +sum u:Time. get(0)@(t+u).P req(pid,t+u,0);
```

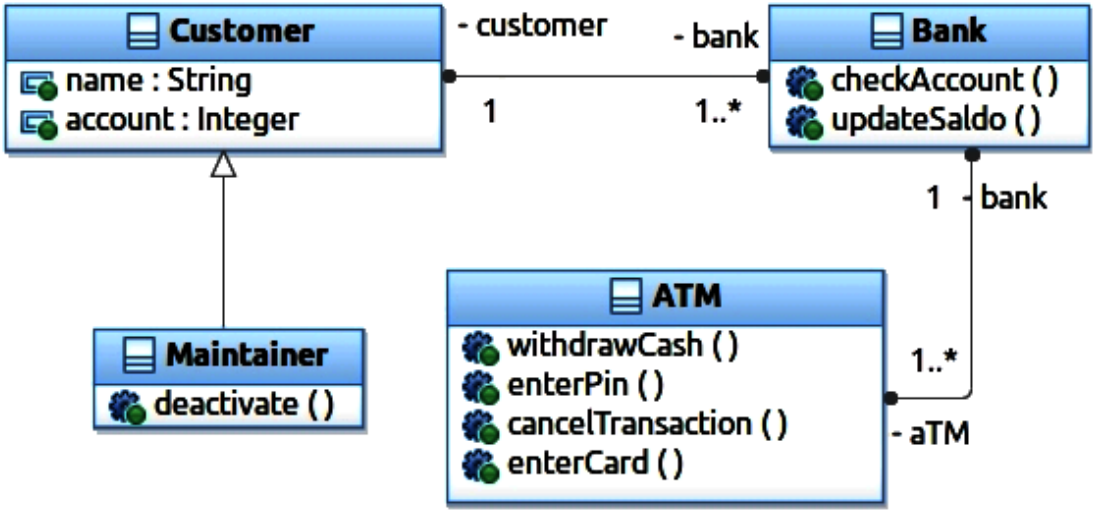
```
<true*>(exists d:D .<ra(d)>
     (nu X. mu Y. (<i_lost>X || <!i_lost && !sb(d)>Y)))
```

```
occur(add_car)
exec(move_lift(rotate))
exec(move_lift(basement))
exec(move_belts([b_r1lift, b_r1b], co
exec(move_lift(street))
occur(add_car)
exec(move_lift(basement))
exec(move_belts([b_r1a, b_r1lift, b_r
```

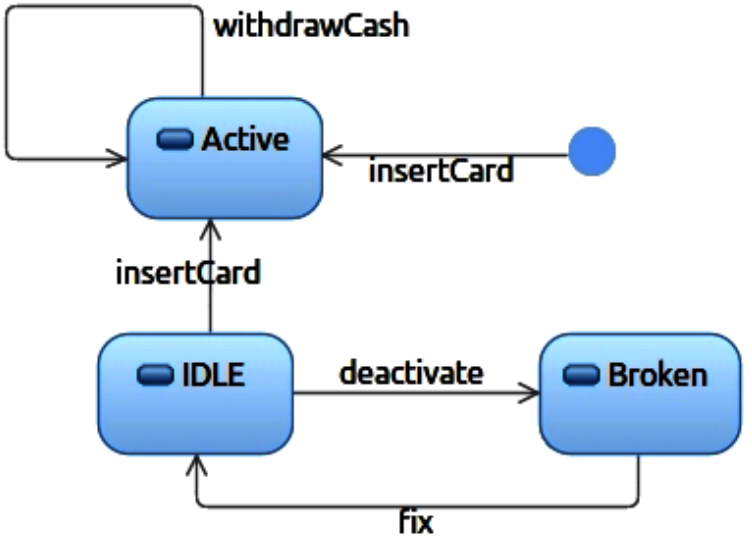
Why *state-of-the-art* is not yet *state-of-the-practice*?

- SE wants efficient push-button verification solutions
  - Not everything is implemented in Java / C / Matlab
  - Not everything is described in a domain-specific verifiable language
- Need to write a funky model in process algebra?
  - Forget it, let's just stick to testing and static analysis.
- UML is the *lingua franca* for describing systems
  - Intuitive, visual, lots of CASE tools, automated test/code generation
  - Not officially formalized!

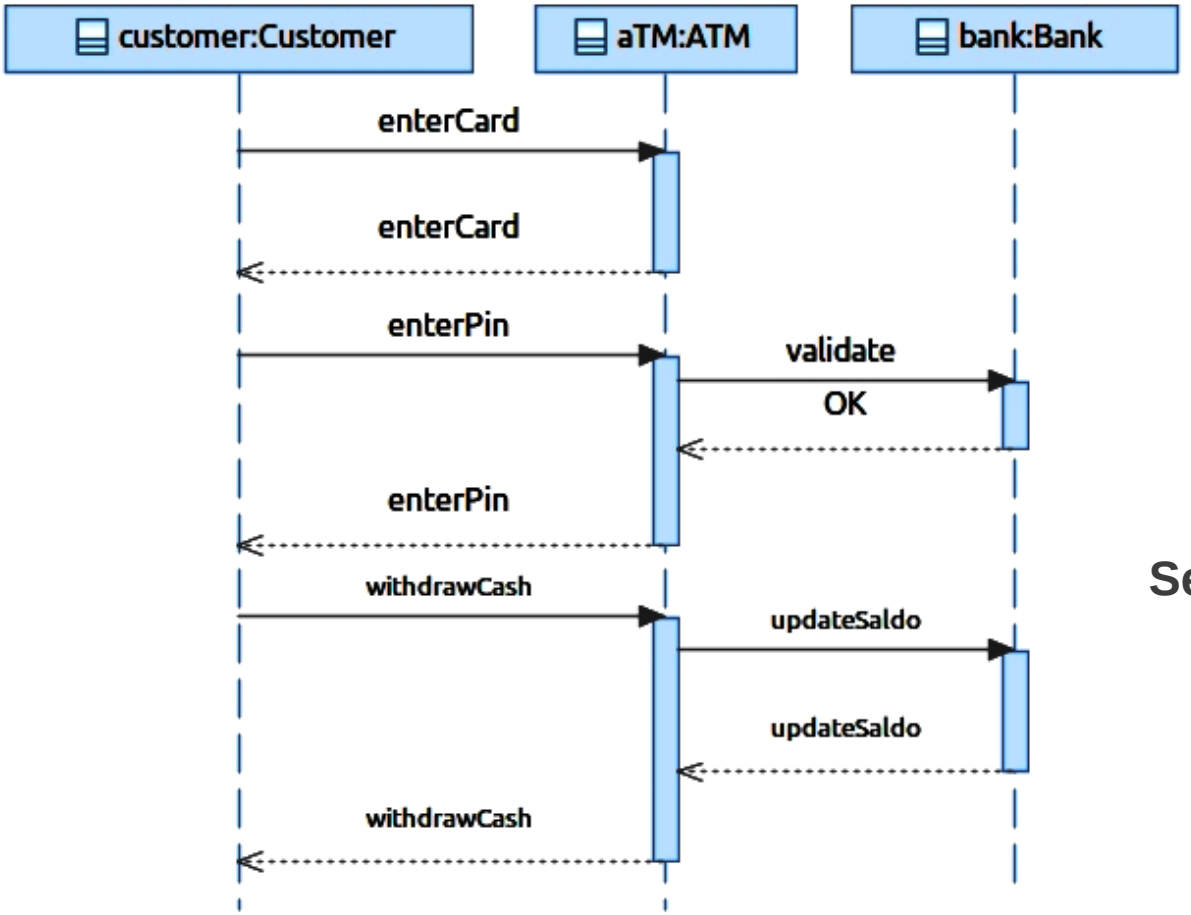
Class Diagram

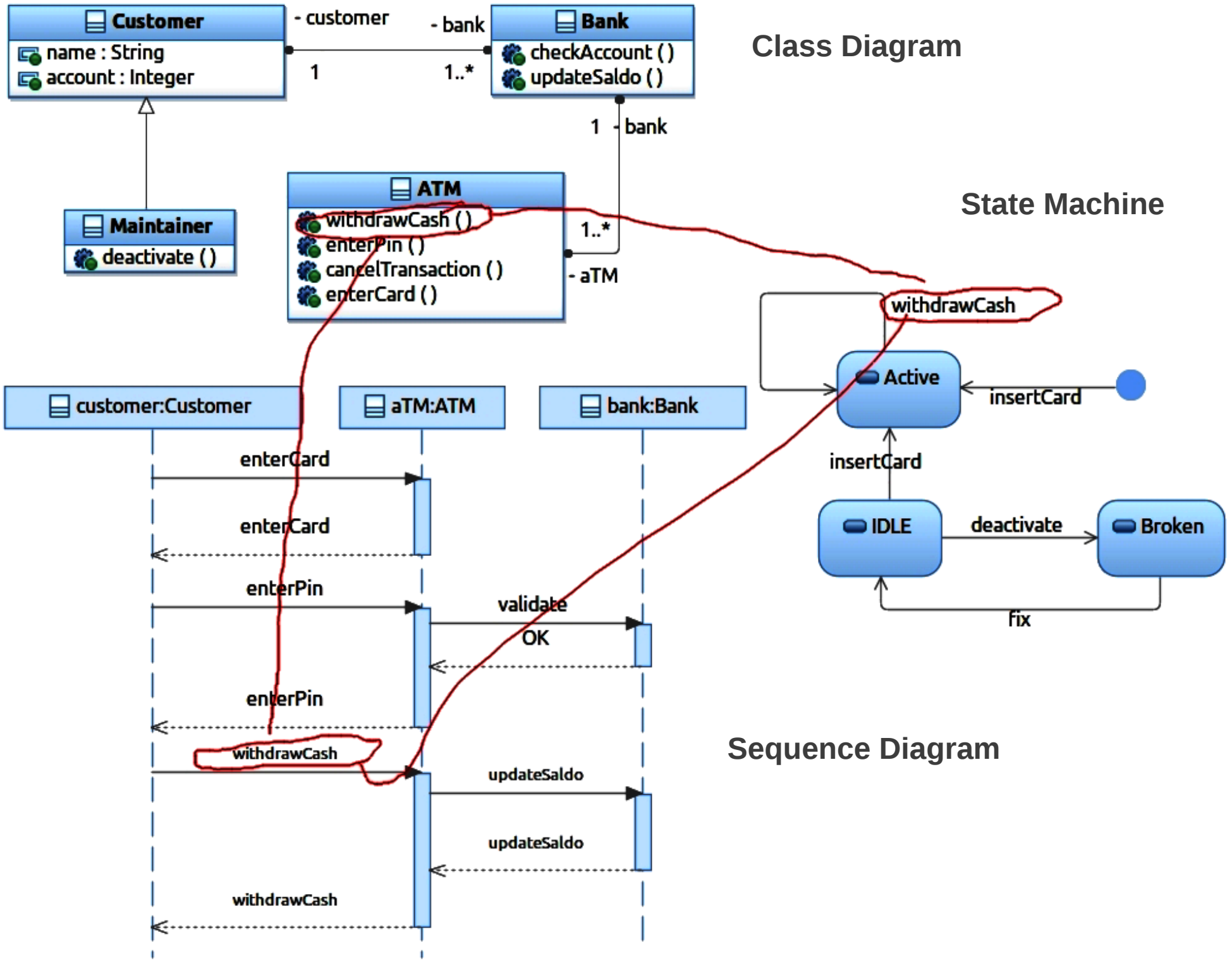


State Machine

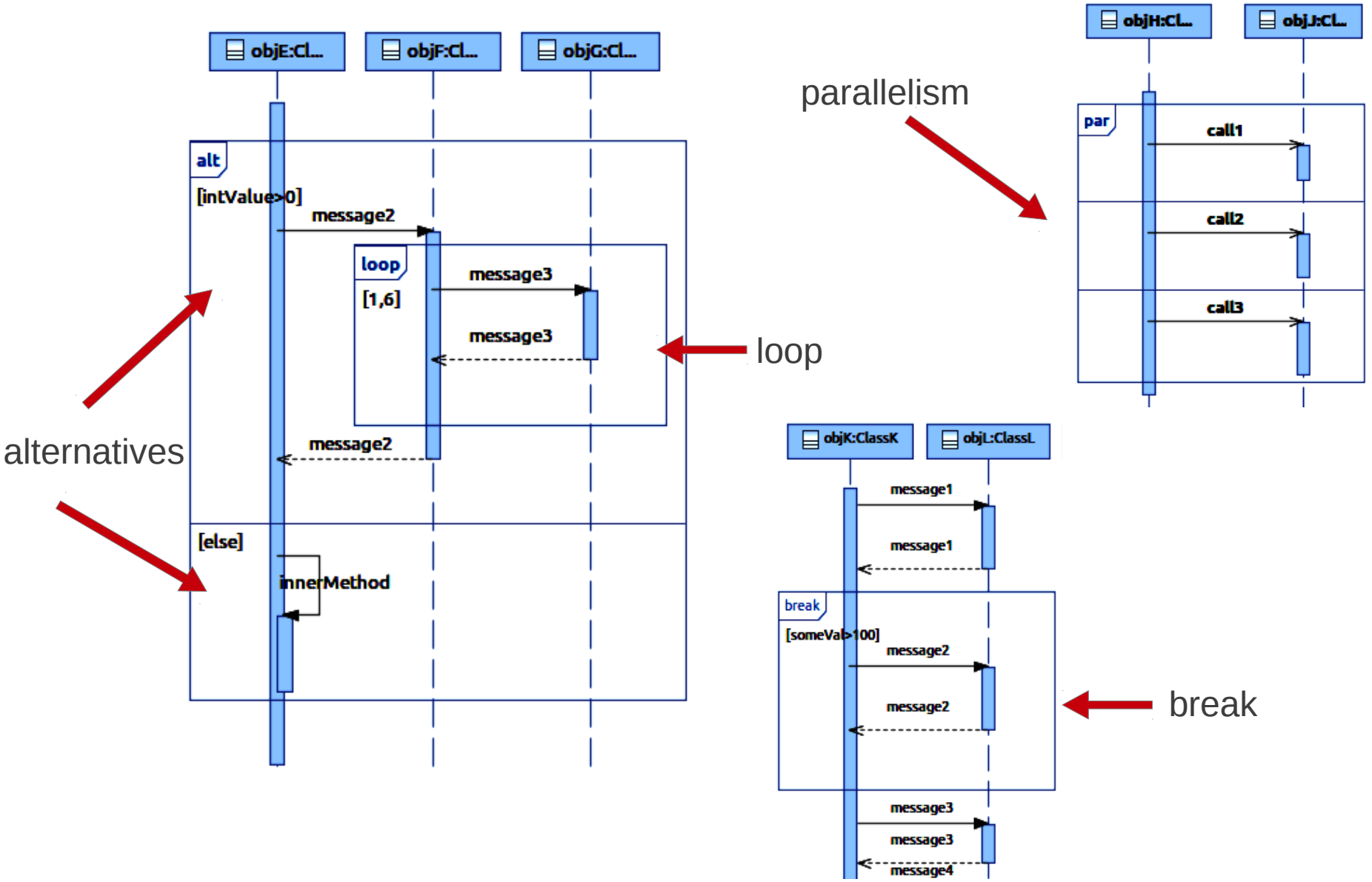


Sequence Diagram

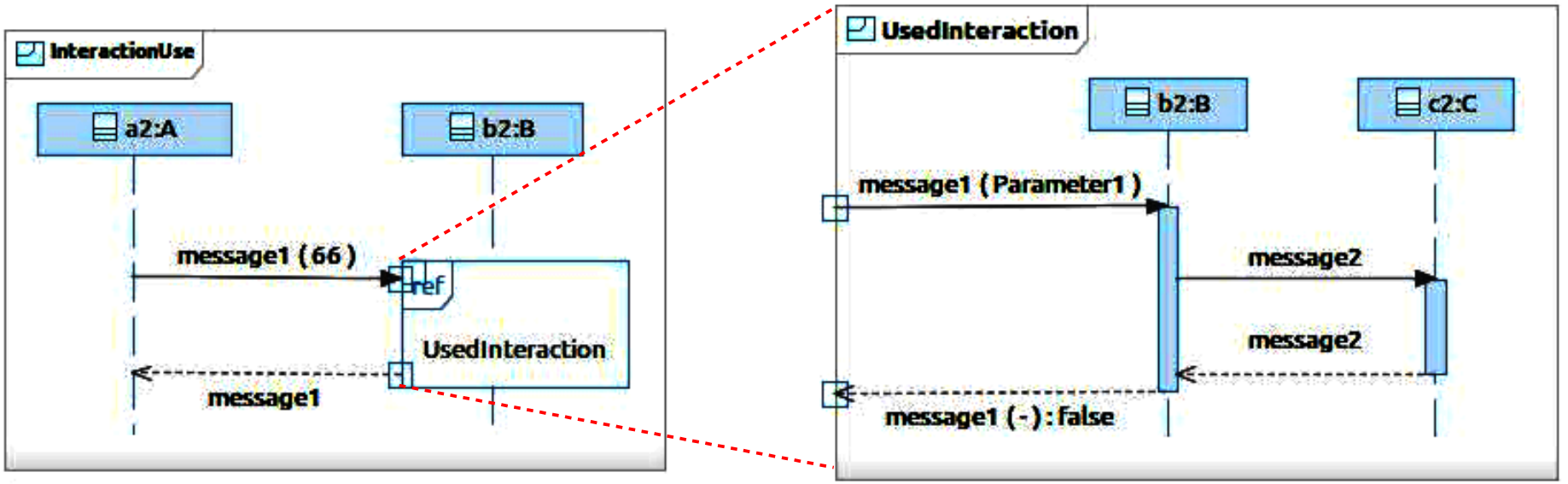



Customer
name : String
account : Integer
- customer
1
- bank
1..*
Bank
checkAccount ( )
updateSaldo ( )
Class Diagram
1
- bank
Maintainer
deactivate ( )
ATM
withdrawCash ( )
enterPin ( )
cancelTransaction ( )
enterCard ( )
1..*
- aTM
State Machine
withdrawCash
Active
insertCard
insertCard
IDLE
deactivate
Broken
fix
customer:Customer
aTM:ATM
bank:Bank
enterCard
enterCard
enterPin
validate
OK
enterPin
withdrawCash
updateSaldo
updateSaldo
withdrawCash
Sequence Diagram

Sequence Diagrams (UML2.x)



Combined Fragments [ref]



For a broad domain of OO software systems, Sequence Diagrams give the most precise description of the behavior, the closest one to code implementation.

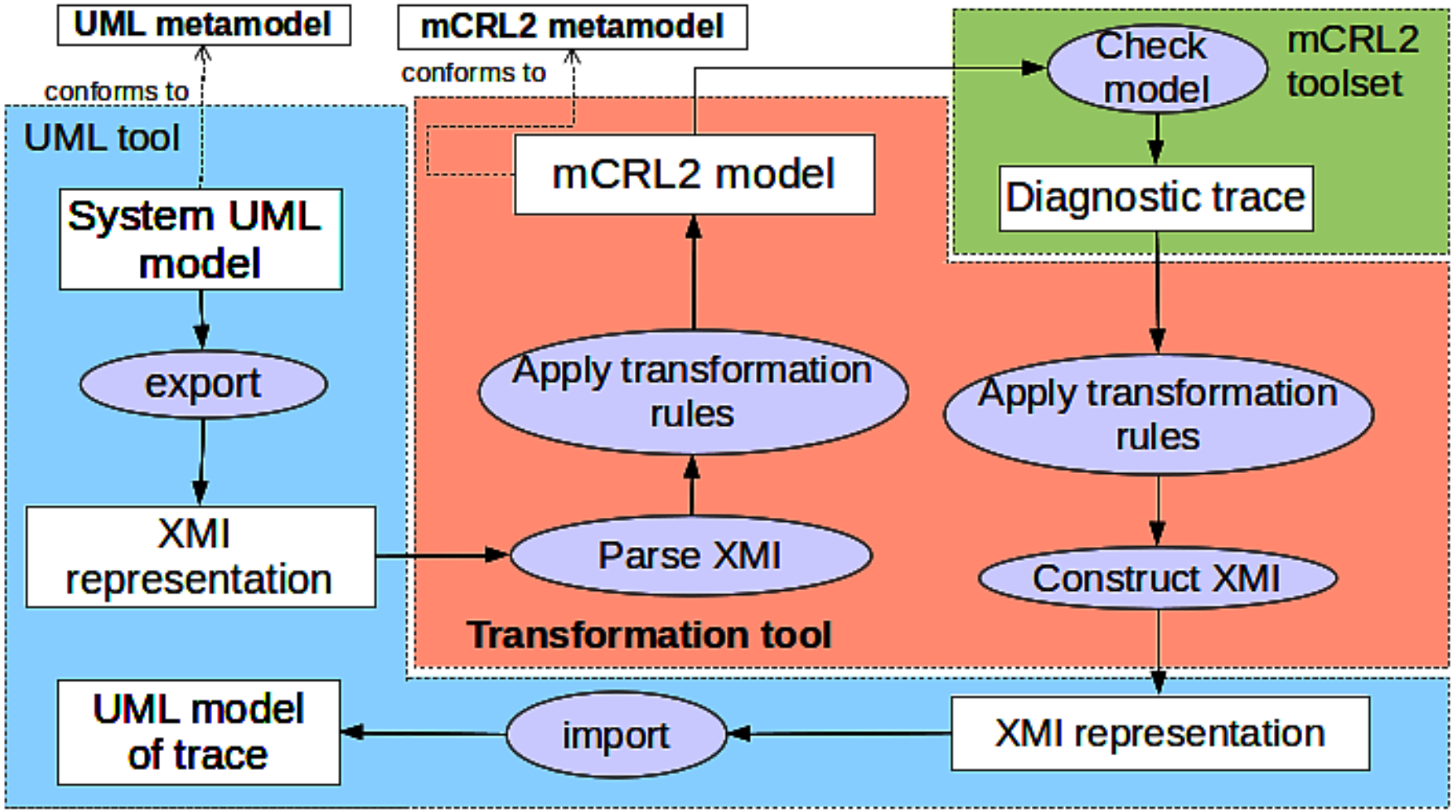
Goal: a full round-trip approach, supported by a toolset.

Ideally, model checking should be hidden from the UML designer!

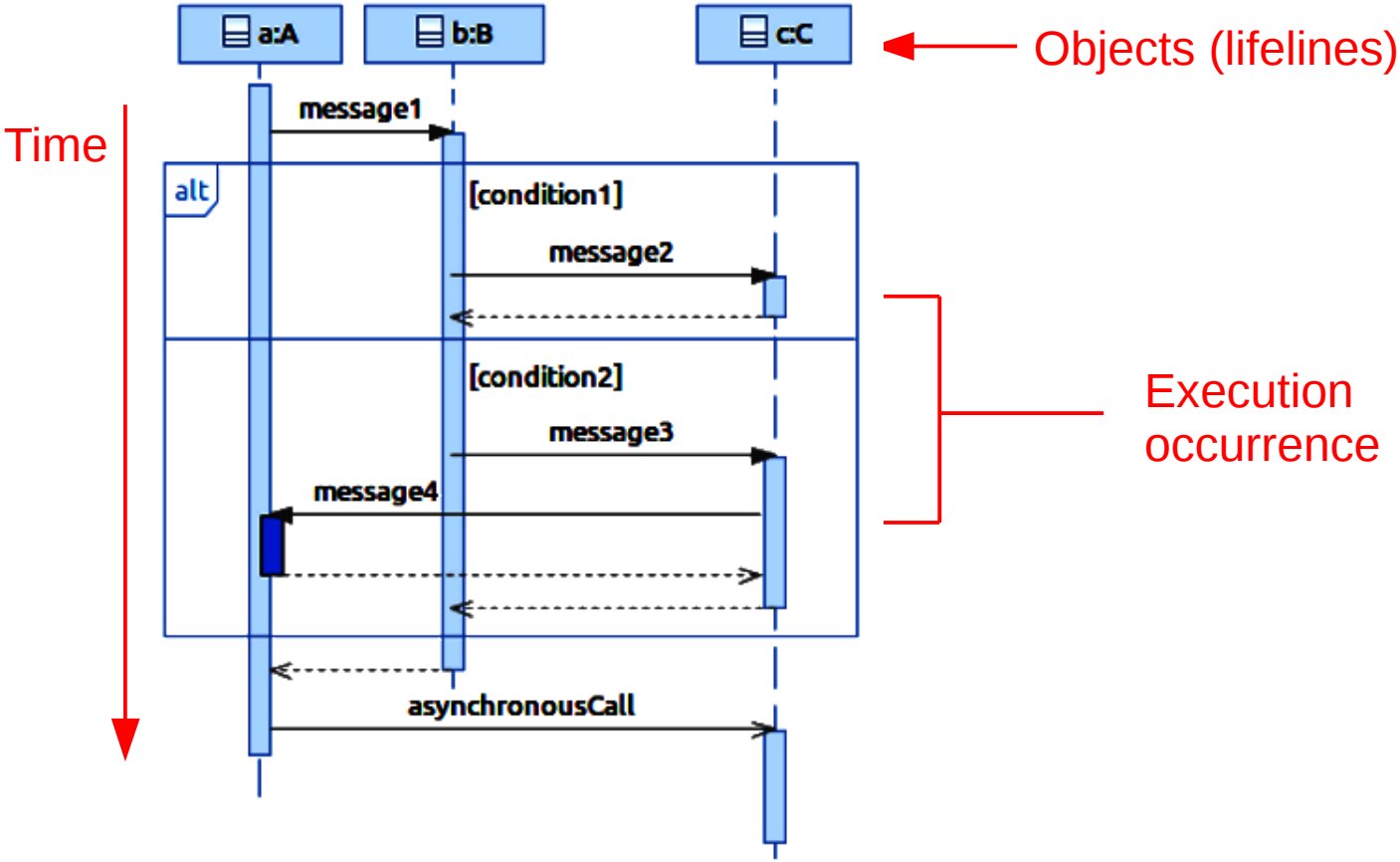
Target formalism: mCRL2

- **actions**: atomic steps
- **processes**: combination (sequential, parallel) of actions …
- **Communication** between processes (exchange of data) via action synchronization ...
- **if-then-else** constructs …
- **nondeterministic** choices …
- means to describe custom **data** structures ... (also:integers,reals,enumerations,booleans,lists,sets..)

The approach



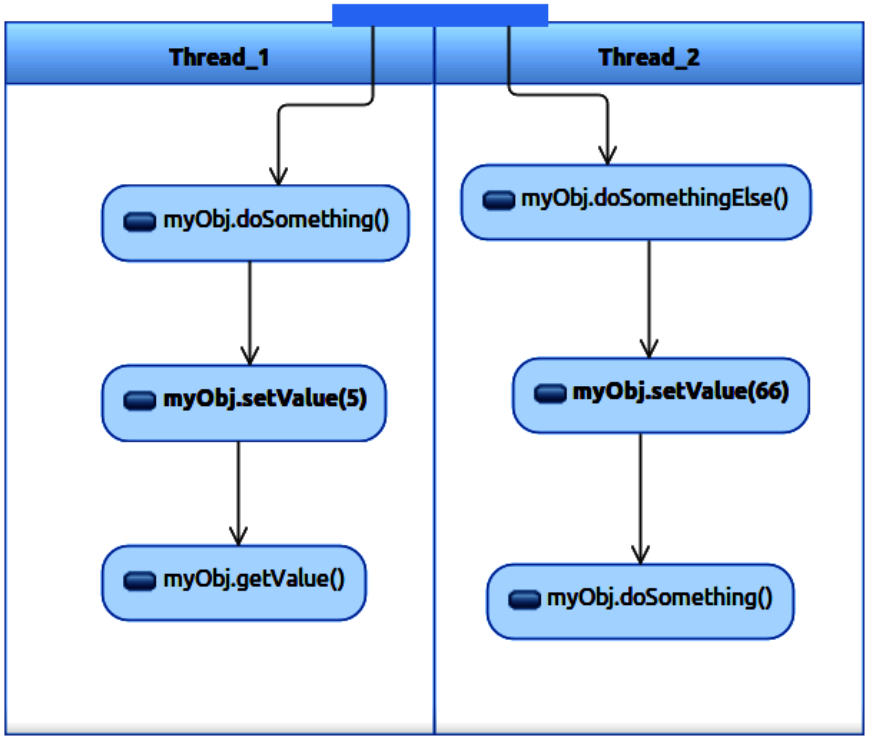
The rationale: treating objects as sequential processes

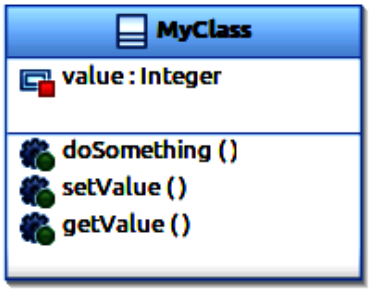


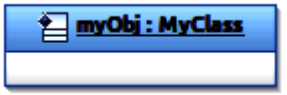
"The general UML-to-Promela formalization approach is to map objects to processes in Spin (proctypes) that exchange messages..."

"...lines 7 and 8 specify the lifelines using process..."

Objects: sequential or concurrent?

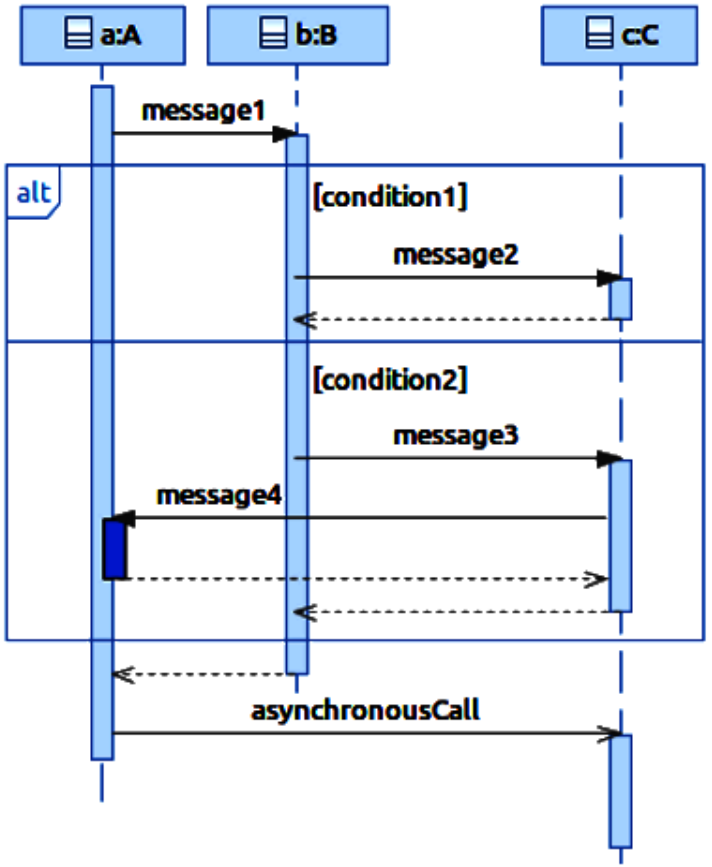






In a concurrent setting, multiple threads of a process could be invoking methods on the same object.

The rationale: global choices



- managing choices globally

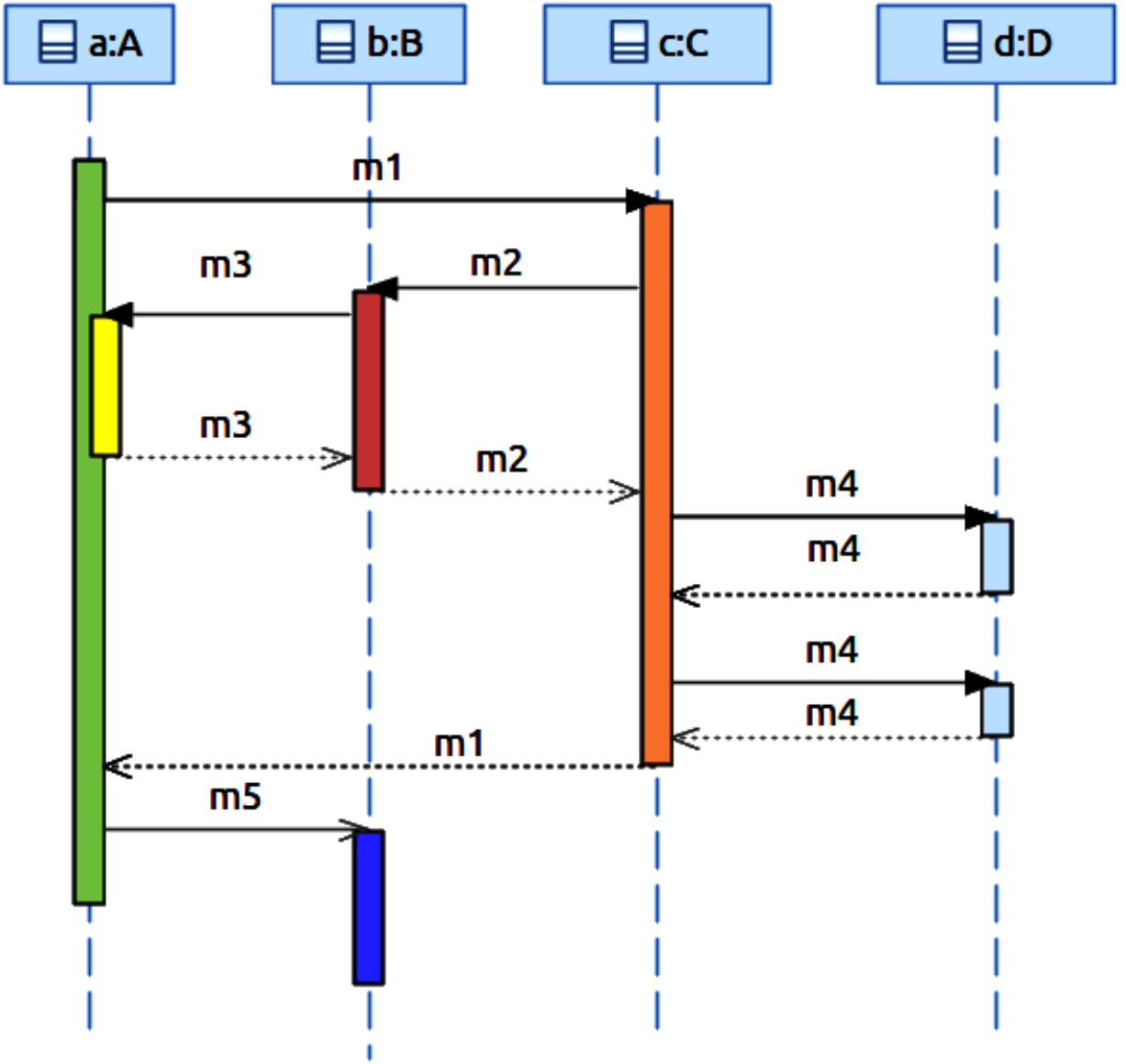
- dealing only with synchronous communication

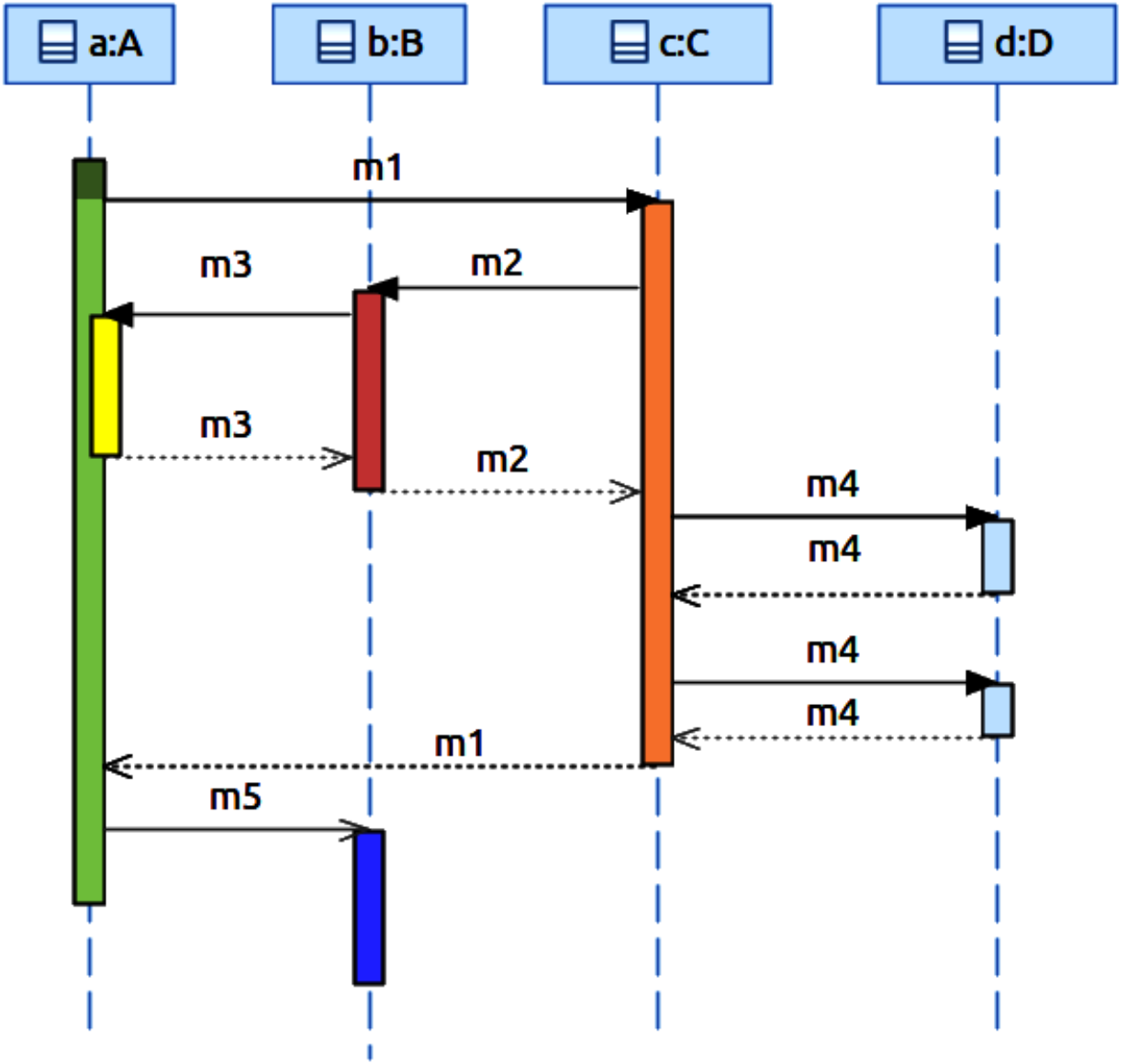
- not treating all Fragment types

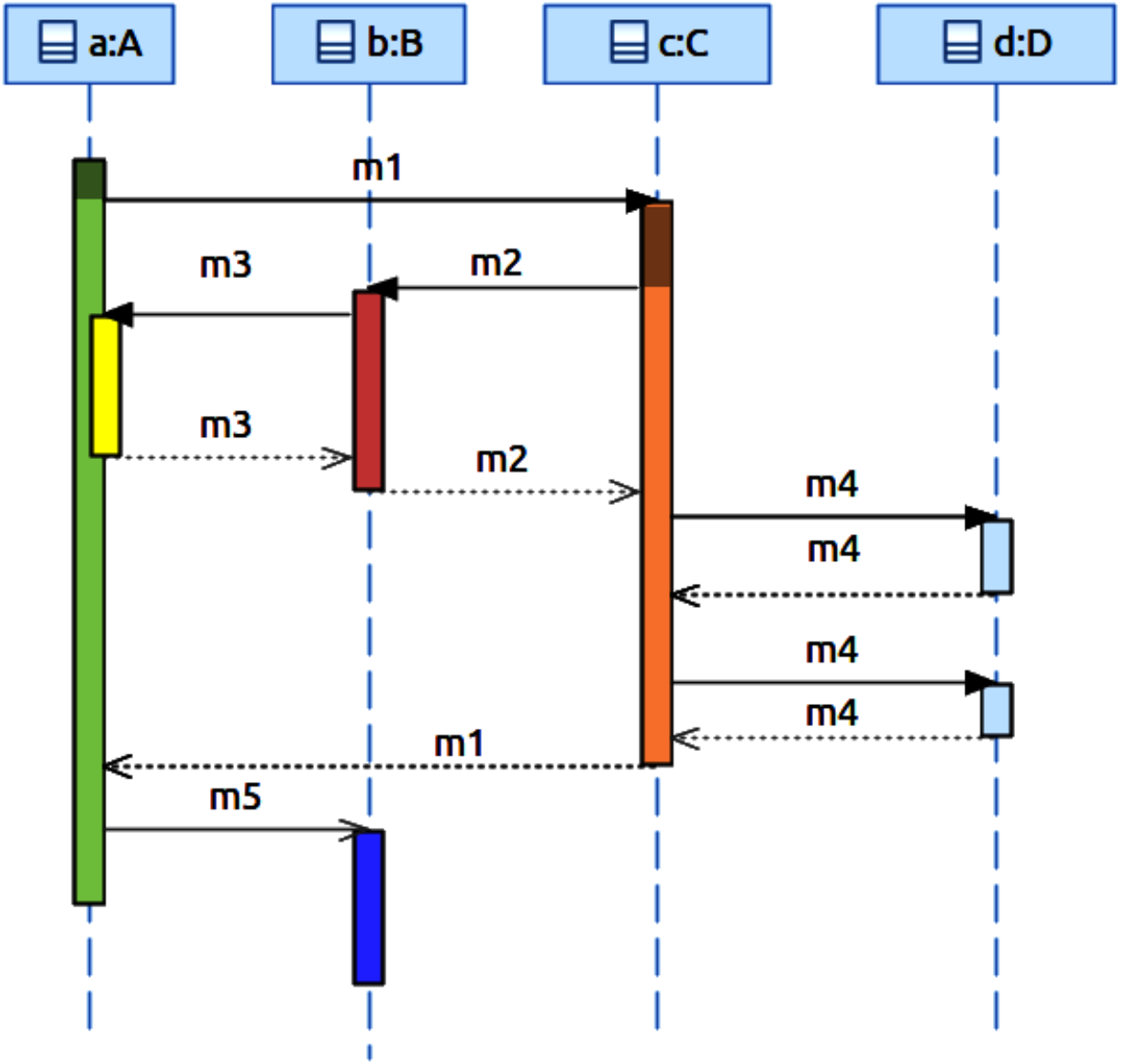
Why should object *a* need to know local decisions of object *b* ?

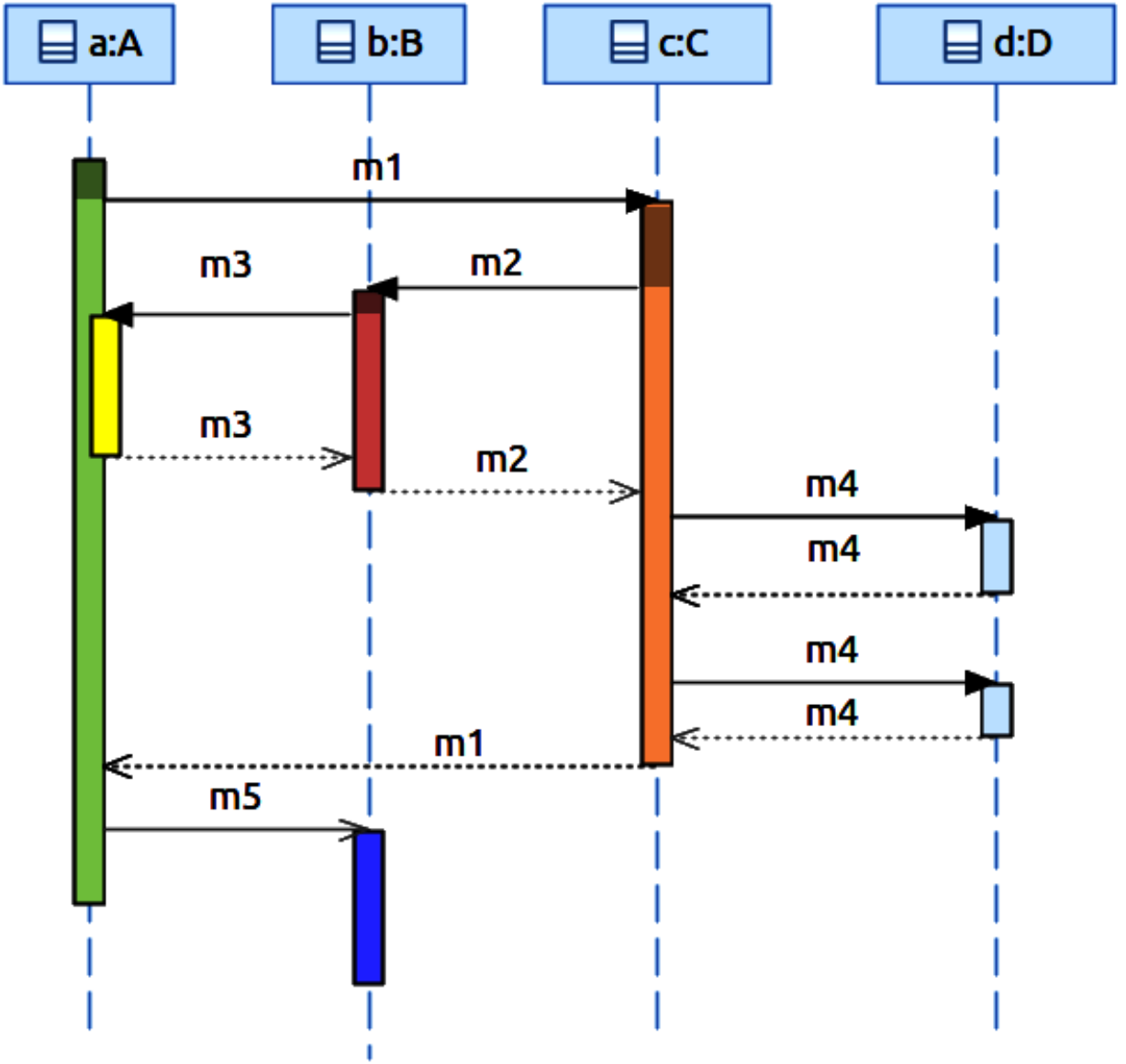
Goal: preserve the OO view in the target model

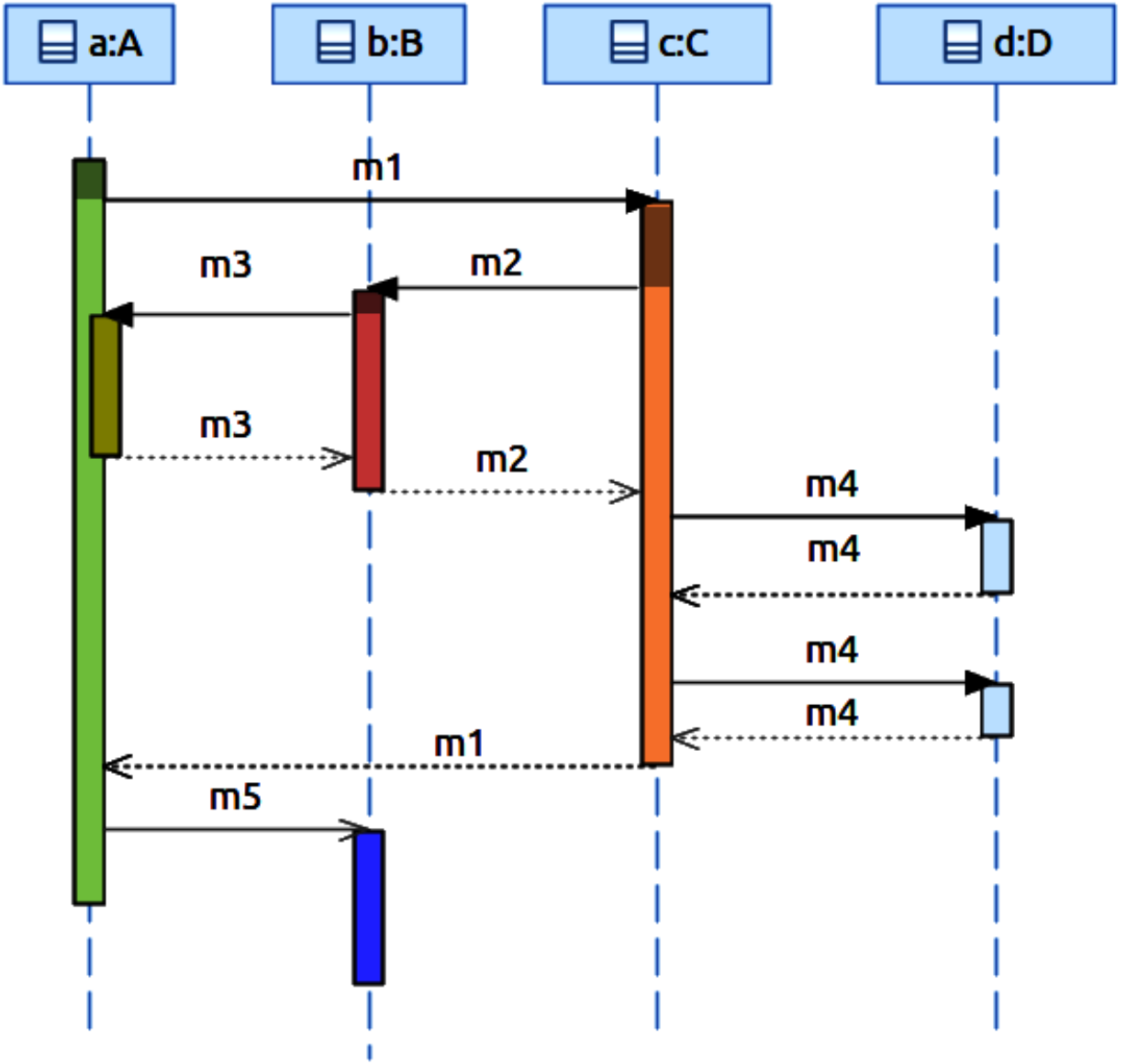
- **An OS level process is essentially a chain of method invocations on objects**
- Associate an mCRL2 ***process description*** with each ***class method***
  - Each mCRL2 process ***instance*** carries information about the class, object, and OS process instance to which the method behavior belongs
- Trivial to reverse model-checking traces back to the UML domain

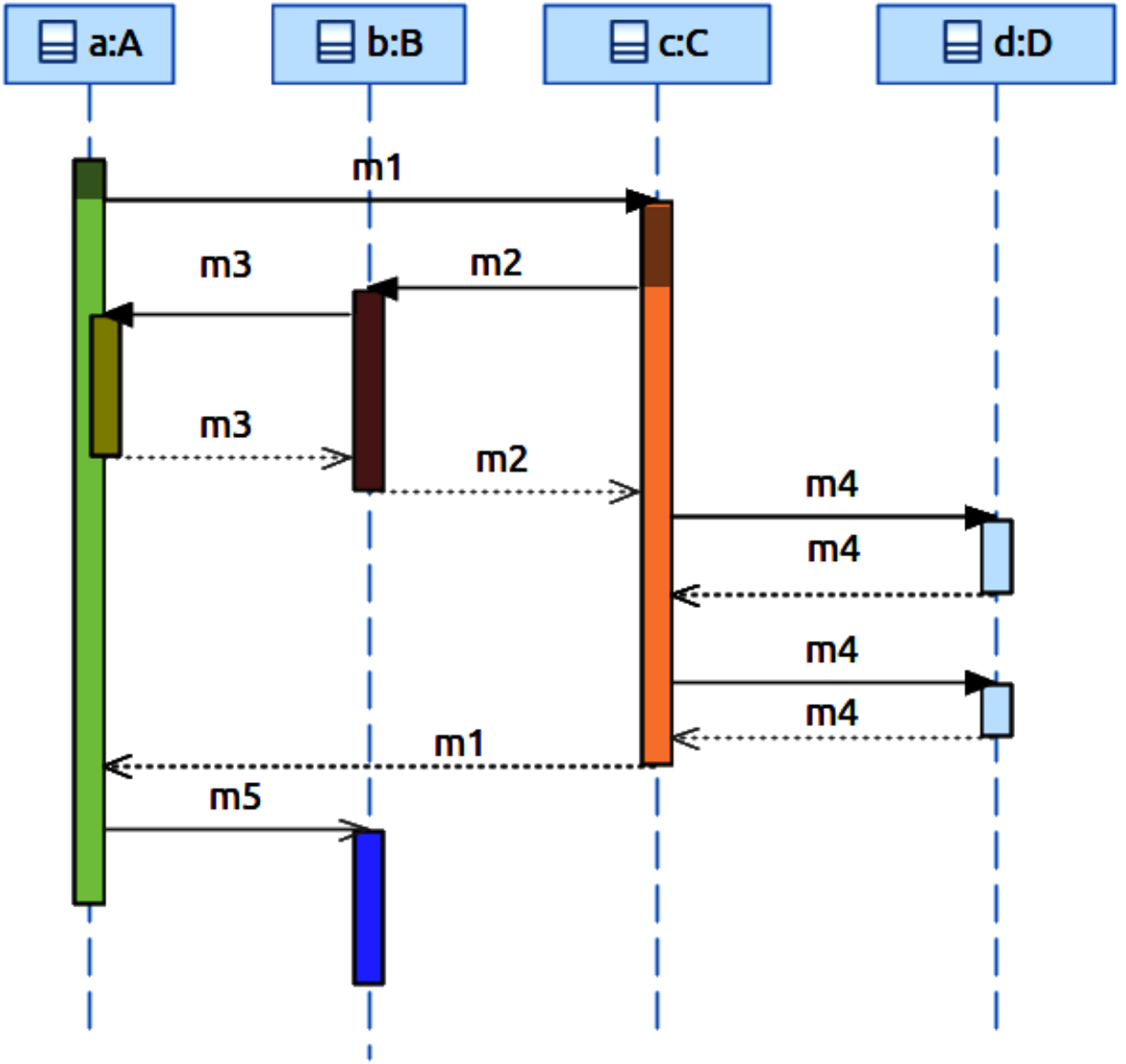

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

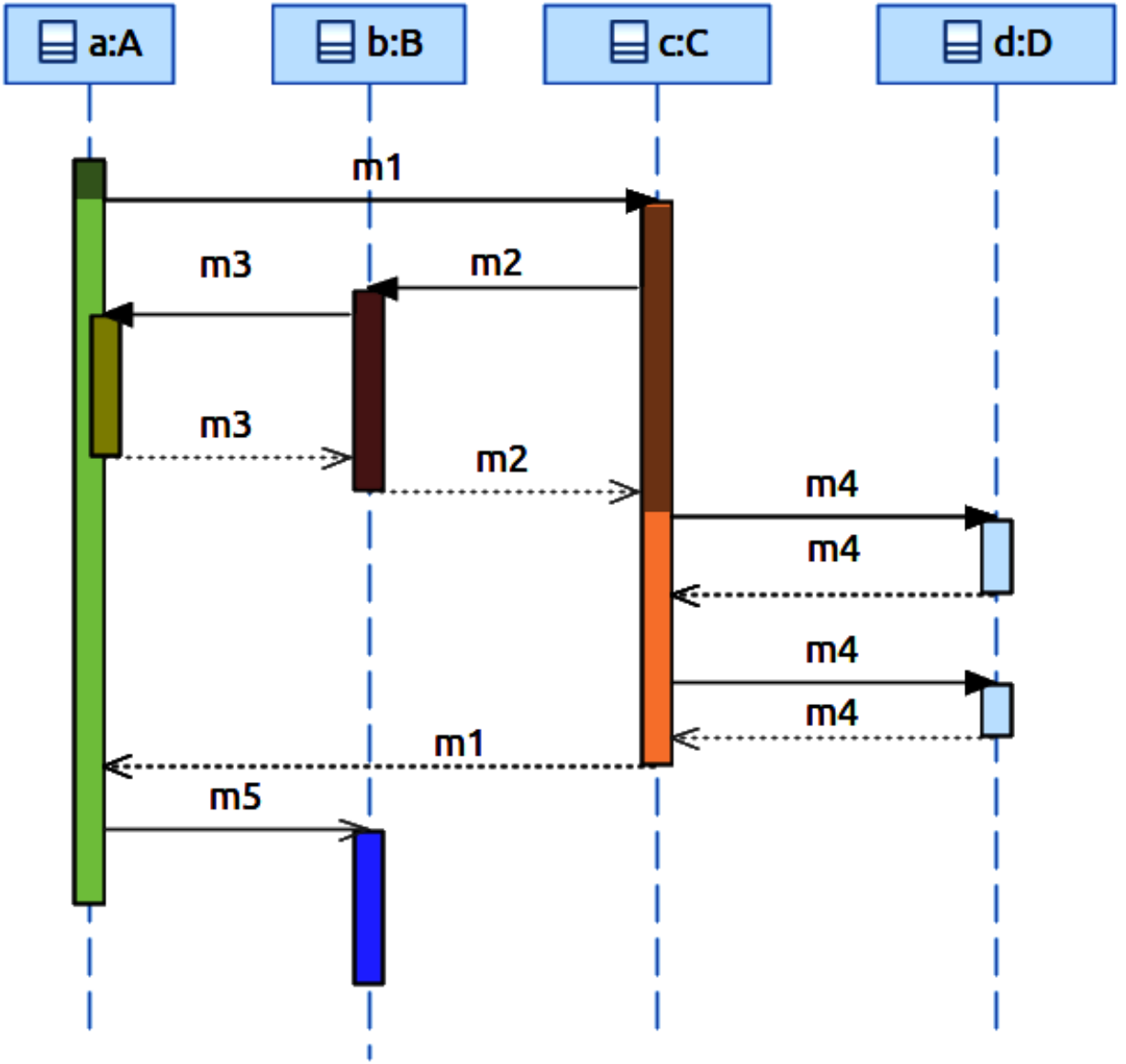

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

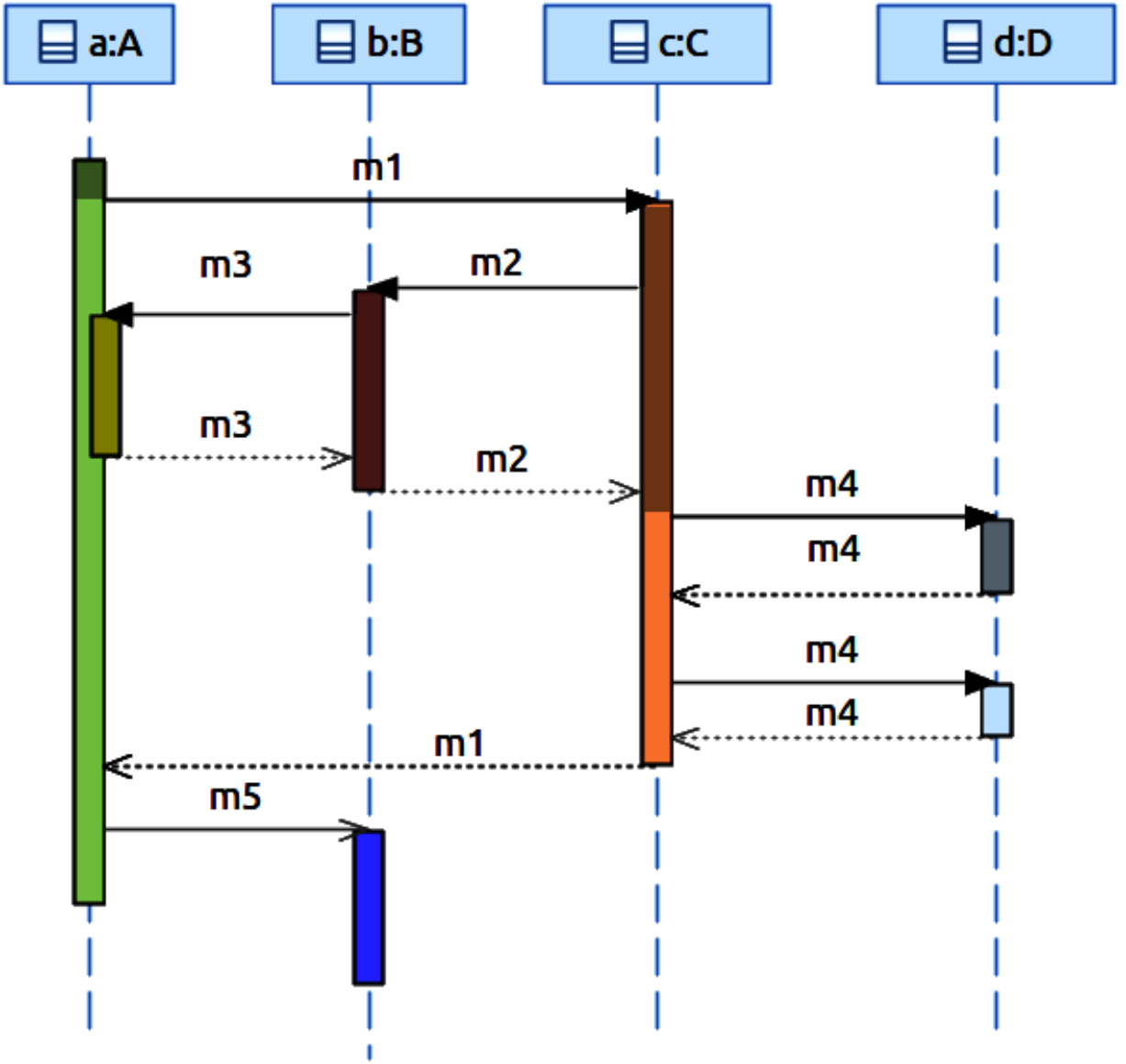

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

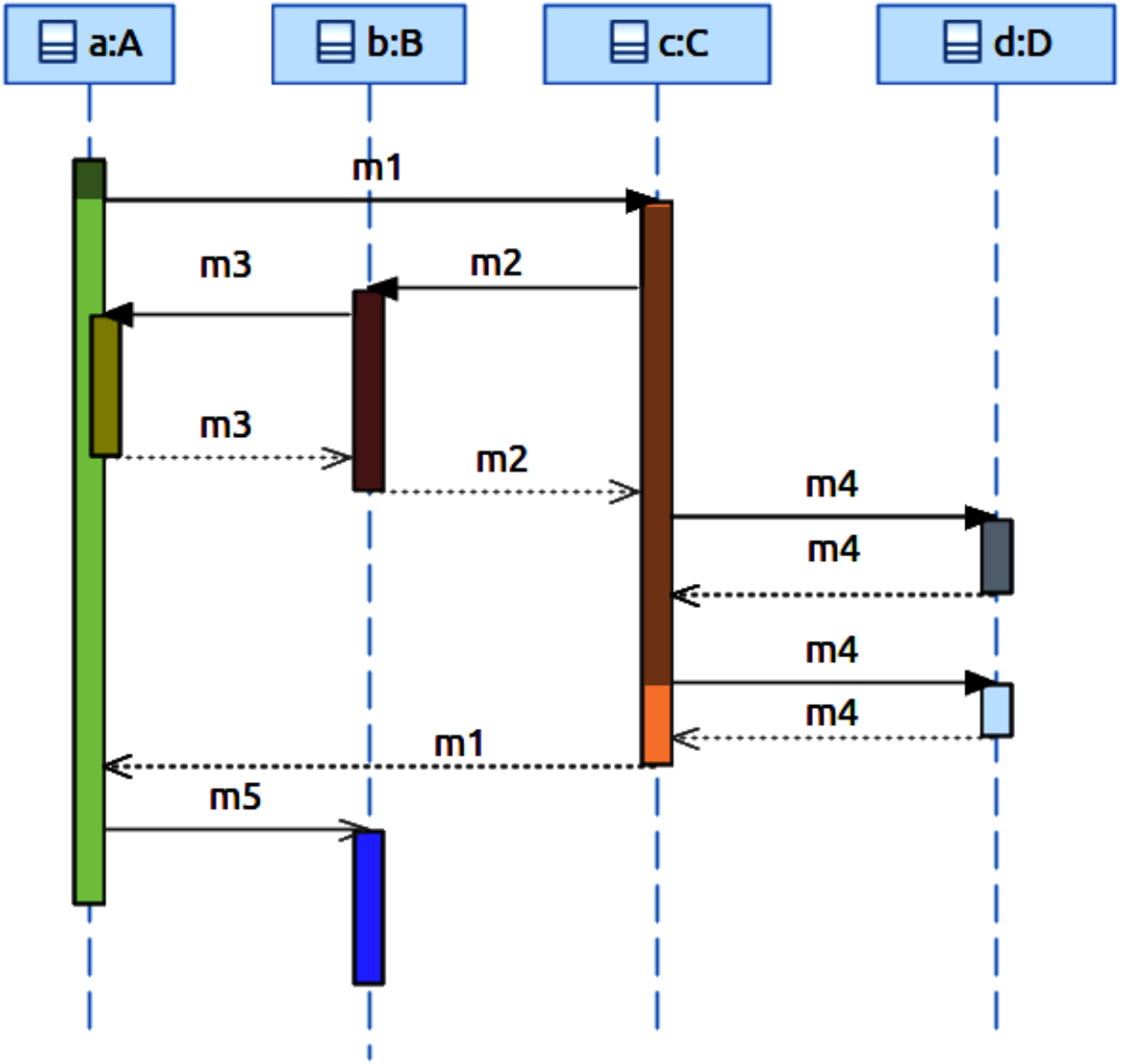

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

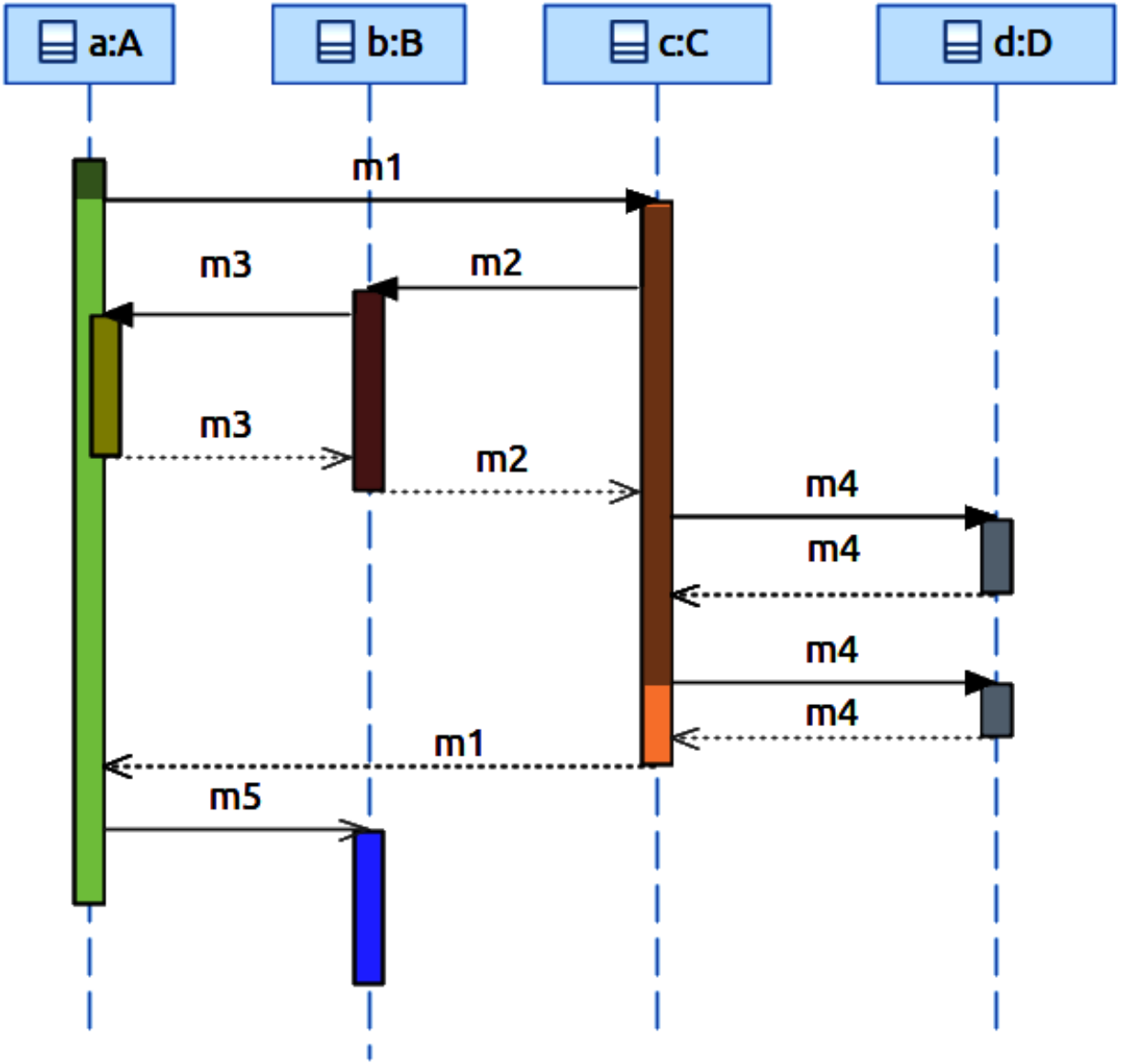

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

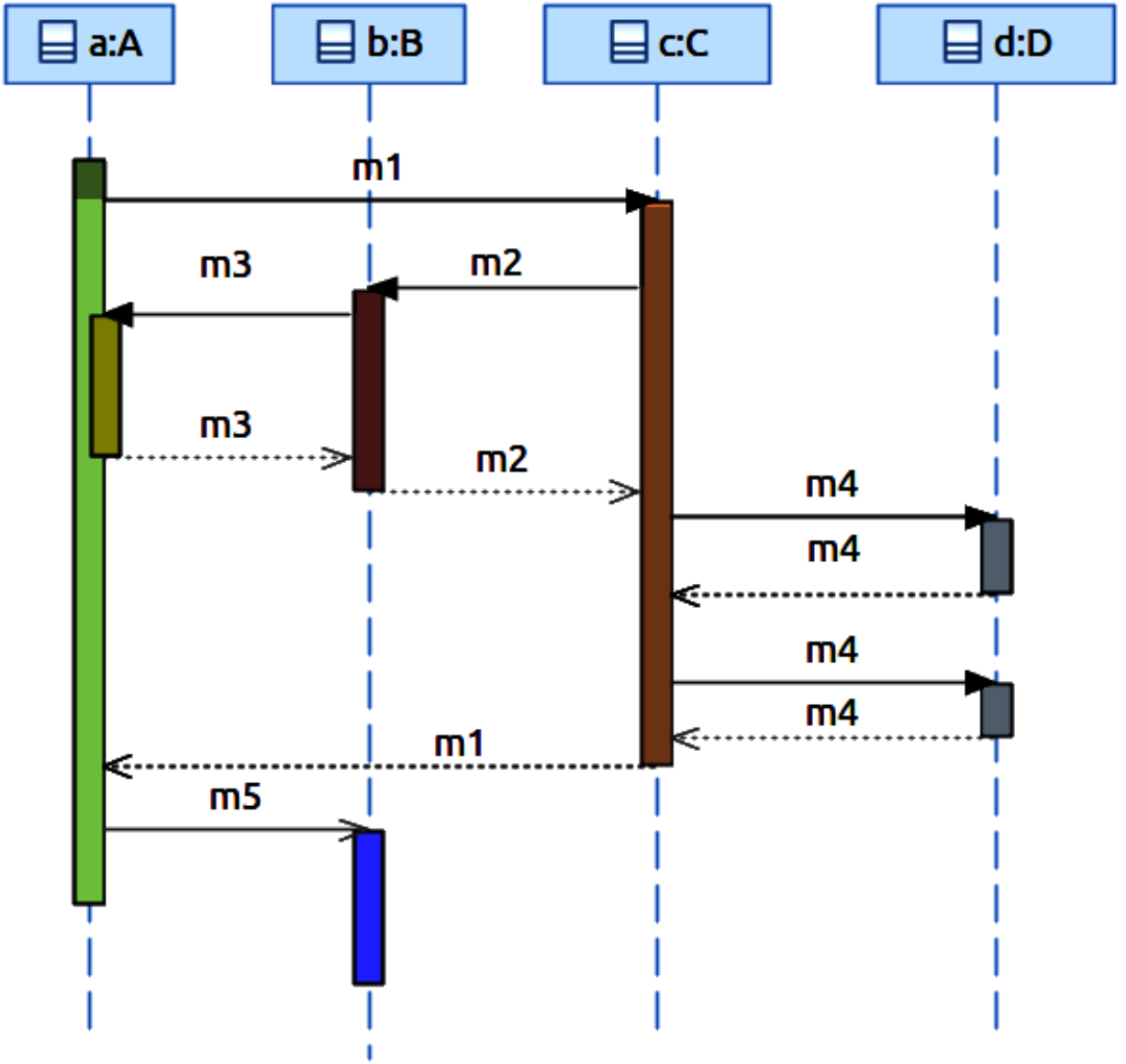

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

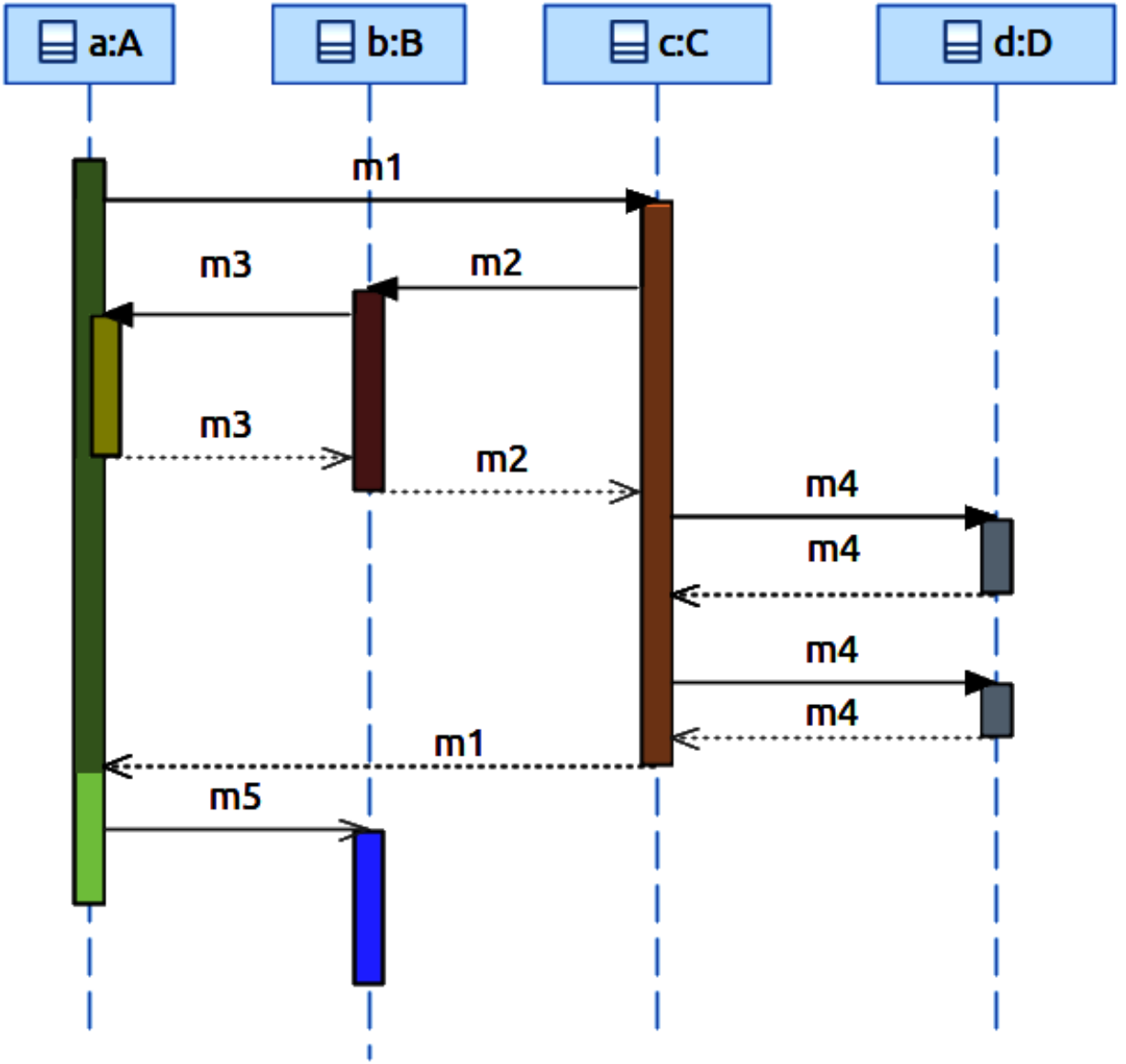

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

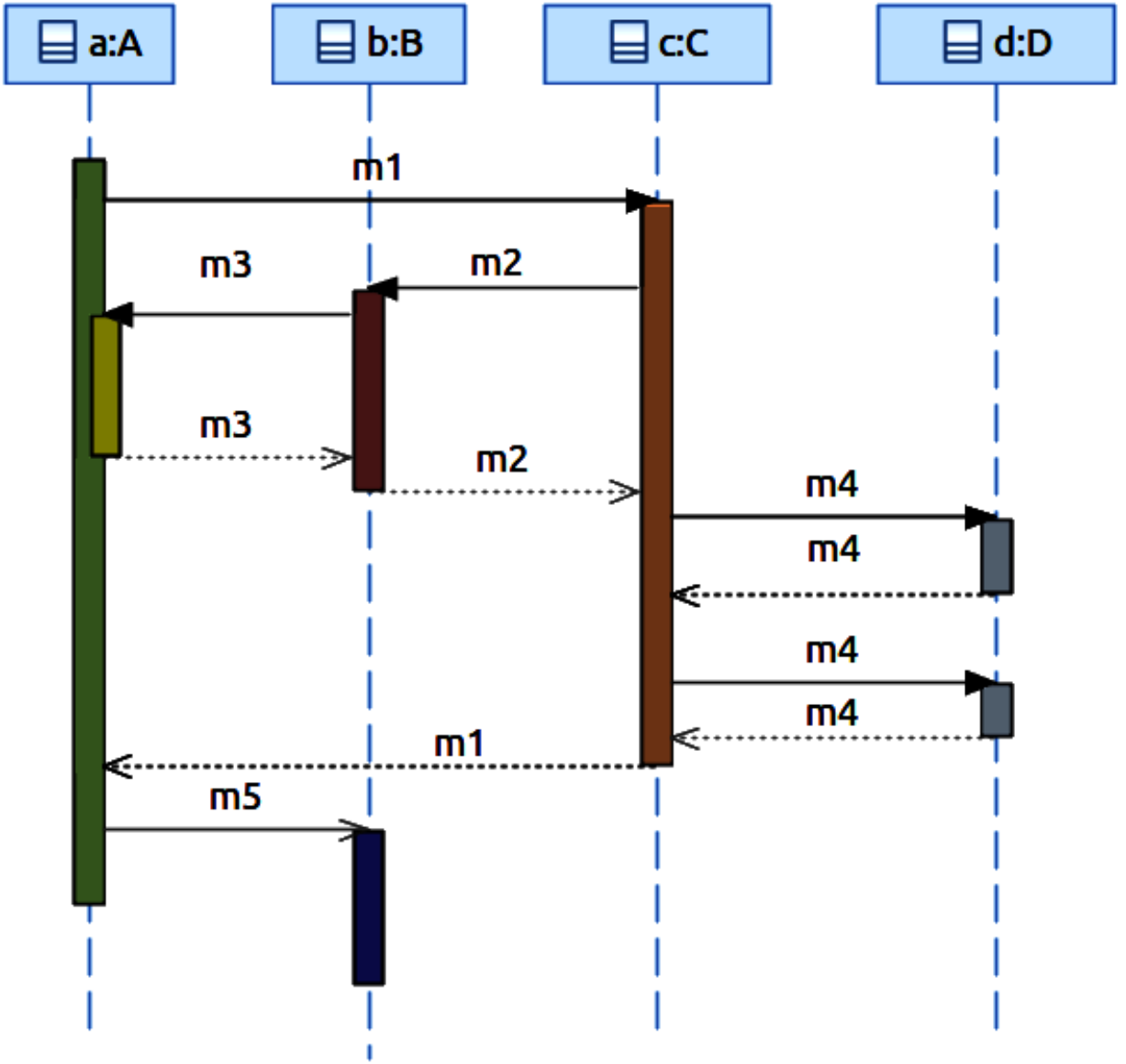

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5

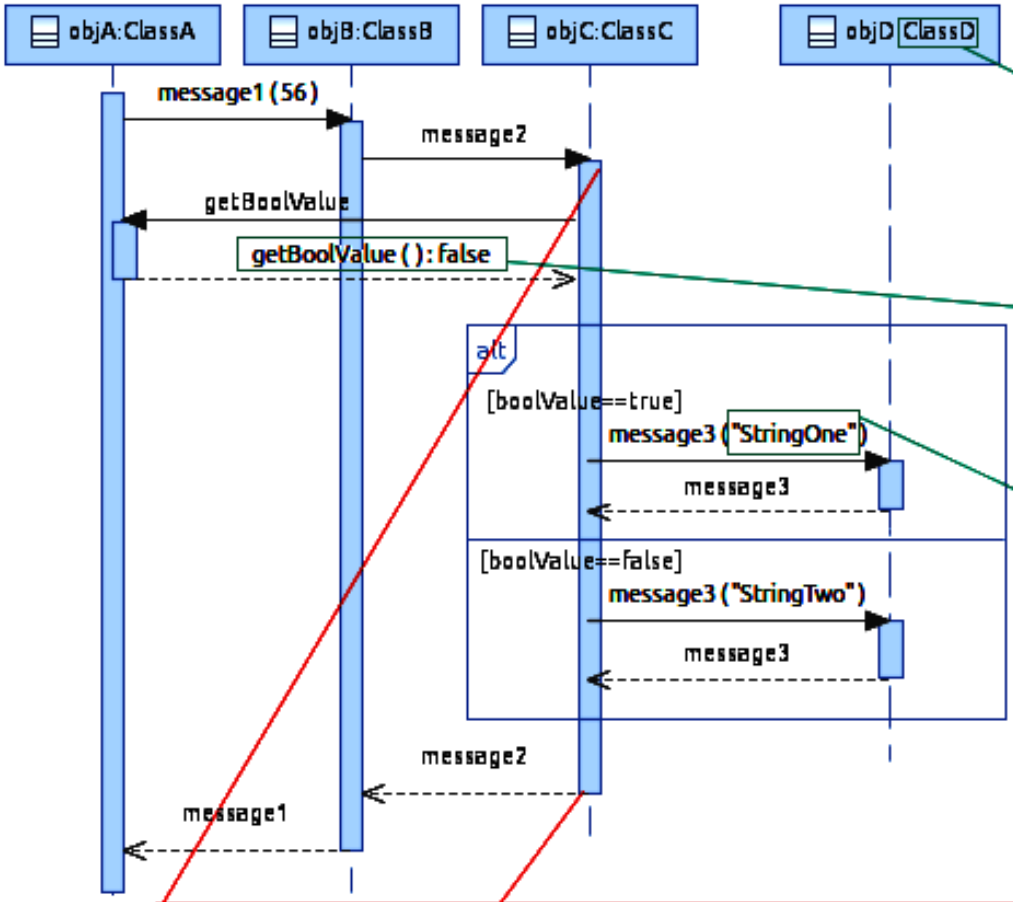

a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5


a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5


a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5


a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5


a:A
b:B
c:C
d:D
m1
m3
m2
m3
m2
m4
m4
m4
m4
m1
m5



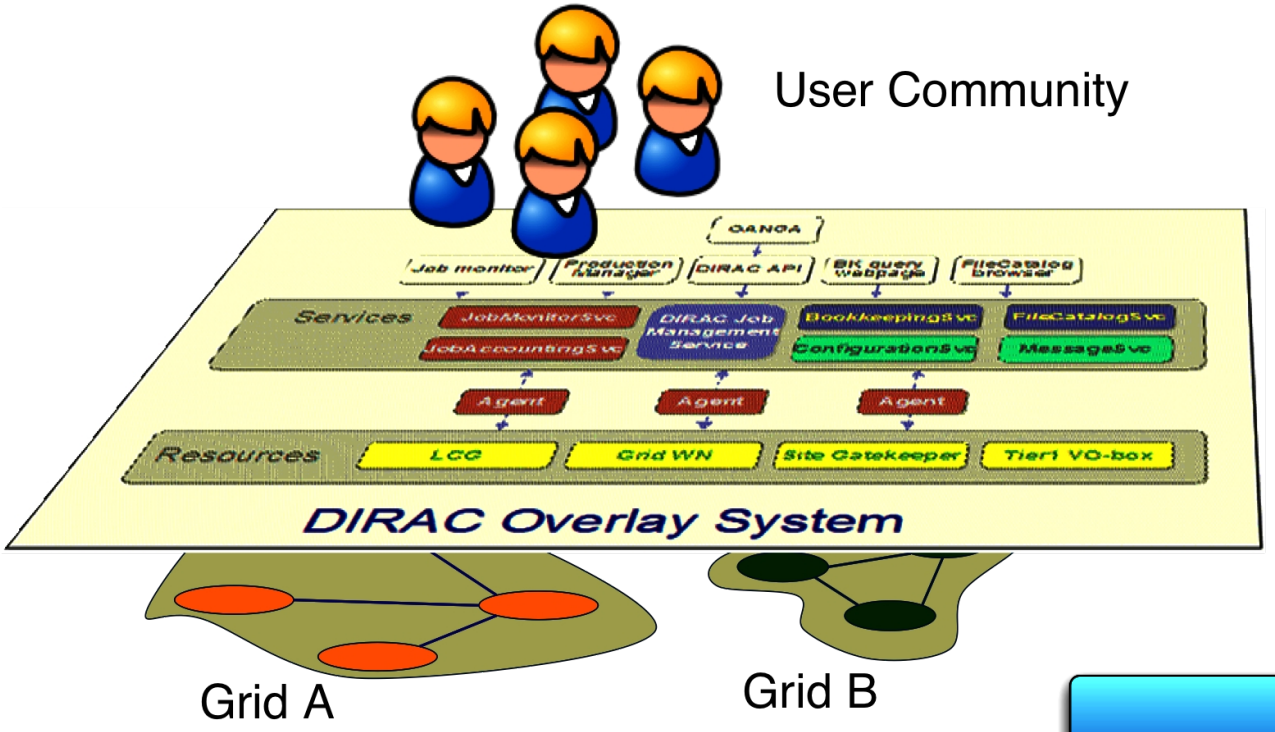

```
sort ClassType = struct
        ClassA |
        ClassB |
        ClassC |
        ClassD ;

sort Method = struct
getBoolValue |
getBoolValue_return(boolValue:Bool) |
message1(intValue:Int) |
message1_return |
message2 | message2_return |
message3(Parameter1:SortString) |
message3_return ;

sort ClassObject = struct objA |
                           objB |
                           objC |
                           objD ;

sort SortString = struct StringOne |
                         StringTwo ;
```

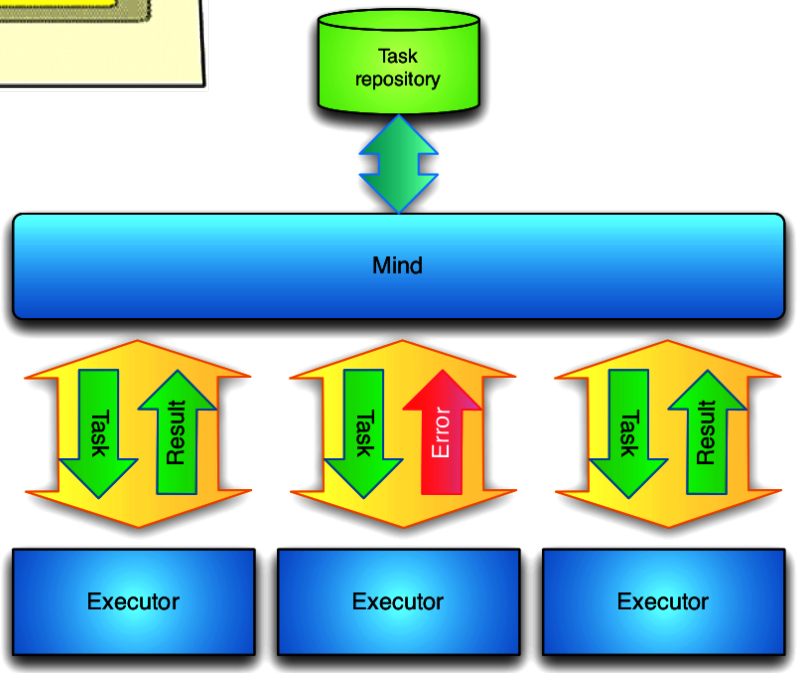
```
proc ClassC_message2(id:Nat) =
sum obj:ClassObject.synch_call_receive(id,ClassC,obj,message2).
synch_call_send(id,ClassA,objA,getBoolValue).
sum boolValue:Bool.synch_reply_receive(id,ClassA,objA,getBoolValue_return(boolValue))
((boolValue==true)->(
        synch_call_send(id,ClassD,objD,message3(StringOne)).
        synch_reply_receive(id,ClassD,objD,message3_return)) <>
(boolValue==false)->(
        synch_call_send(id,ClassD,objD,message3(StringTwo)).
        synch_reply_receive(id,ClassD,objD,message3_return)
) <> internal).
synch_reply_send(id,ClassC,obj,message2_return);
```

Validation

- UML specification is semi-formal
  - Semantics deduced via partial meta-model views & natural language descriptions
  - No mathematically-formalized semantics
  - We don't have formal correctness proofs to support the validity of this transformation
  - Simulation on simple building blocks; application on a real case study

Case Study: DIRAC Executor Framework

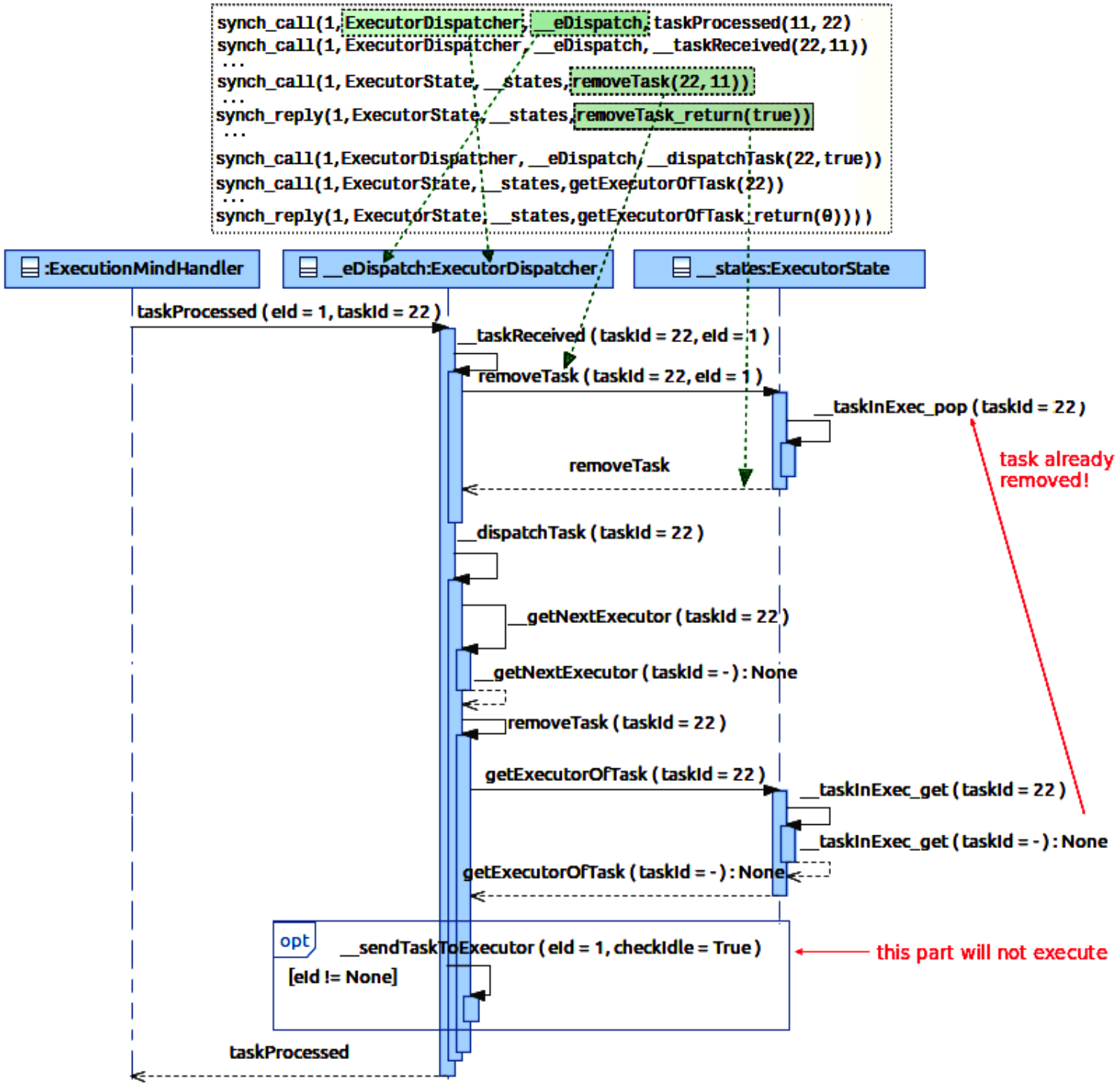




```
$ java -jar UML2mCRL2 exportedModel.uml model.mcrl2
$ mcrl22lps -nfbw model.mcrl2 model.lps
$ lps2pbes model.lps model.pbes –formula=formula.mcf
$ pbes2bool model.pbes
```

```
[true* .
synch_call(1,ExecutorQueues,_queues,pushTask(JobPath,taskId,false)).
true*.
!(synch_call(1,ExecutorQueues,_queues,popTask([JobPath])))*.
synch_reply(1,ExecutorDispatcher,_eDispatch,
_sendTaskToExecutor_return(OK,0))]false
```

- Problem: state-space explosion!
  - 50 processes in the model
  - >300million states and >600GB of memory
- Workaround: standard monitoring automaton running lock-step with the model, fires a deadlock action if a violation is found


synch_call(1, ExecutorDispatcher, __eDispatch, taskProcessed(11, 22))
synch_call(1, ExecutorDispatcher, __eDispatch, __taskReceived(22,11))
...
synch_call(1, ExecutorState, __states, removeTask(22,11))
...
synch_reply(1, ExecutorState, __states, removeTask_return(true))
...
synch_call(1, ExecutorDispatcher, __eDispatch, __dispatchTask(22,true))
synch_call(1, ExecutorState, __states, getExecutorOfTask(22))
...
synch_reply(1, ExecutorState, __states, getExecutorOfTask_return(0))))
:ExecutionMindHandler
__eDispatch:ExecutorDispatcher
__states:ExecutorState
taskProcessed (eId = 1, taskId = 22)
__taskReceived (taskId = 22, eId = 1)
removeTask (taskId = 22, eId = 1)
__taskInExec_pop (taskId = 22)
removeTask
task already removed!
__dispatchTask (taskId = 22)
__getNextExecutor (taskId = 22)
__getNextExecutor (taskId = -) : None
removeTask (taskId = 22)
getExecutorOfTask (taskId = 22)
__taskInExec_get (taskId = 22)
__taskInExec_get (taskId = -) : None
getExecutorOfTask (taskId = -) : None
opt
__sendTaskToExecutor (eId = 1, checkIdle = True)
[eId != None]
this part will not execute
taskProcessed

Conclusions and Future Work

- Goal: bridge the existing gap by providing transformation methodology and toolset to verify UML models

- **Express properties in UML rather than with μ-calculus**

  remember this?

```
[true* .
synch_call(1,ExecutorQueues,_queues,pushTask(JobPath,taskId,false)).
true*.
!(synch_call(1,ExecutorQueues,_queues,popTask([JobPath])))*.
synch_reply(1,ExecutorDispatcher,_eDispatch,
_sendTaskToExecutor_return(OK,0))]false
```